Programming Project 3: The Barbershop of EVILLLLLLL!

CS415 - Operating Systems

1 Overview

Welcome to Dr Lewis's Barbershop of Evil! Our barbershop has a single barber and a single barber's chair. It also has a finite number of chairs for waiting customers. Our barber, while evil, is lazy: they will nap in the barber's chair if there are no customers waiting. What can we do to make certain the barber is working when there are customers, sleeping when there are none, and do so with a properly evil order?

2 Background

This is a classic synchronization problem in operating systems.

So... one barber's chair means that the barber finishes cutting a customer's hair, that customer pays and then the barber looks to see if there are customers waiting for a haircut. If so, he puts the customer who first arrived into the barber's chair and cuts their hair. If there are none, the barber takes a nap in his barber chair.

When a customer arrives, they look to see what the barber is doing at that time. If they are sleeping, they wake them up and the customer gets their hair cut. If the barber is cutting hair, the customer checks to see if there is an available seat. If there is a seat, they take the seat and wait their turn for a haircut. If no seat is available, the customer leaves and goes down the street to Great Clips.

The problem here is that a number of actions take a random amount of time: - The barber requires a random amount of time to do a haircut - The customer takes a random amount of time to get to a waiting-room chair (customers do like to talk).

This is a problem for mutexes and condition variables (used as semaphores). The room status forms a critical section: the barber must acquire a room status mutex before checking for customers and release the mutex when they begin to sleep or cut hair.

A customer must acquire the room status mutex when they enter the shop and release it once they sit in a waiting room chair, or they sit in the barber chair, or if they leave if no seats are available.

3 Problem statement

Simulate this problem using two threads, two mutexes, and a condition variables.

3.1 Synchronization variables

- A mutex that indicates the barber is ready (barberReady), unlocked at initialization and locked while the barber is cutting hair.
- A mutex that protects updates to the number of available waiting seats (waitChairsCanBeAccessed), locked at initialization.
- A condition variable that keeps track of the number of customers in the waiting room (readyCustomers), initialize to zero.
- Also need to keep a count of the number of available customer seats.

3.2 The algorithm:

```
def Barber():
    LOOP
        See if customer has arrived (readyCustomers unlocked & > 0)
        Awake - acqure the customer chair mutex
        Increment the number of available customer seats
        Unlock the barber ready mutex
        Unlock the customer chair mutex
        Cut hair for a random amount of time
    END LOOP

def Customer():
   LOOP
        Try to lock the mutex protecting updates to number of chairs
        IF number of available seats > 0 THEN
           decrement the number of available customer seats
           Signal the readyCustomers condition variable
           Unlock the mutex protecting access the chairs
           Wait until the barber is ready
           Get your hair cut
        ELSE
           Unlock the mutex protecting acces to the chairs
           Leave the barbershop disappointed
        ENDIF
   END LOOP
```

Your task... implement this algorithm in C++ using STL threads. Run your implementation for a simulated amount of time (measured by using a counter starting at zero up to a randomly generated integer between 100 and 1000). Once the program completes, print the number of customers serviced by the barber and the number of customers who left the shop.

4 Submitting your program

Your submission document must be in PDF format; submission of documents in any other format will result in deduction of points from your grade. Combine together source code, examples of your program running, and test data into a single PDF file. Attach your submission to the assignment entry in Blackboard.